BIO392: Introduction to R

Hangjia Zhao
hangjia.zhao@uzh.ch
2025.04.10

What is R

R is an integrated suite of software facilities for data manipulation, calculation and graphical display.

- Based on a another programming language (called ’S’)
- Works on different platforms (Windows, Mac, Linux)
- Open-source and free
- It has a large community support and has been extended by a large collection of packages (libraries of functions) that can be used to solve different problems.

Basic set up

Install R

go to https://cloud.r-project.org/

RStudio

an integrated development environment (**IDE**) : a software application that helps programmers develop software code efficiently

Install: https://posit.co/download/rstudio-desktop/

Install R packages

- from The Comprehensive R Archive Network (CRAN)

```
install.packages("package_name")
```

- from Bioconductor

```
install.packages("BiocManager")
BiocManager::install("package_name")
```

- from GitHub

```
install.packages(“devtools”)
devtools::install_github(“username/repo_name”)
```

Using functions: get help with help pages by using `? (e.g. ?sqrt)`

R variables

Variables are containers for storing data values.

- A variable is created the moment you first assign a value to it.
- To assign a value to a variable, use the <- sign. To output (or print) the variable value, type the variable name or use print() function.

Example:

```
course <- “BIO392”

course # auto-print the value of the course variable

print(course) # print the value of the course variable
```

In other programming language, it is common to use = as an assignment operator. In R, we can use both = and <- as assignment operators. However, <- is preferred in most cases because the = operator can be forbidden in some context in R.

R operators

Operators are used to perform operations on variables and values.

- Arithmetic operators: x+y

  + (Addition) - (Subtraction) * (Multiplication) / (Division) ^ (Exponent) %% (Remainder from division) %/% (Integer Division)

- Assignment operators: x <- 1

  <- <<- (a global assigner)

- Comparison operators

  == (Equal) != (Not equal) > (Greater than) < (Less than) >= (Greater than or equal to) <= (Less than or equal to)

- Logical operators

  & Element-wise Logical AND operator. It returns TRUE if both elements are TRUE

  && Logical AND operator - Returns TRUE if both statements are TRUE

  | Element-wise Logical OR operator. It returns TRUE if one of the elements is TRUE

  || Logical OR operator. It returns TRUE if one of the statement is TRUE.

  ! Logical NOT - returns FALSE if statement is TRUE

- Miscellaneous operators

  : Creates a series of numbers in a sequence e.g. x <- 1:10

  %in% Find out if an element belongs to a vector e.g. x %in% y

  %*% Matrix Multiplication e.g. x <- Matrix1 %*% Matrix2

R data types

- numeric: 10.5, 55, 787
- integer: 1L, 55L, 100L, where the letter "L" declares this as an integer
- complex: 9 + 3i, where "i" is the imaginary part
- character (a.k.a. string): "k", "R is exciting", "FALSE", "11.5"
- logical (a.k.a. boolean): TRUE or FALSE

We can use the class() function to check the data type of a variable

Example:

```
# integer
x <- 1000L
class(x)
```

Conditional execution & loops

If statements

- if

```
a <- 33
b <- 200
if (b > a) {
  print("b is greater than a")
}
```

- else if

```
a <- 33
b <- 33
if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print ("a and b are equal")
}
```

- if else

```
a <- 200
b <- 33
if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

- AND OR condition

```
a <- 200
b <- 33
c <- 500
if (a > b & c > a) {
  print("Both conditions are true")
}
```

Conditional execution & loops

For / While loop

While loop

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

- break

```
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
  if (i == 4) {
    break
  }
}
```

- next

```
i <- 0
while (i < 6) {
  i <- i + 1
  if (i == 3) {
    next
  }
  print(i)
}
```

For loop

```
for (x in 1:10) {
  print(x)
}
```

support break and next statements

R functions

To create a function, use the function() keyword:

```
my_function <- function(x) { # create a function with the name my_function
   5 * x
}
my_function(3) # call the function named my_function with the argument x equals to 3
```

To let a function return a result, use the return() function:

```
my_function <- function(x) { # create a function with the name my_function
   return(5 * x)
}
```

R data structures

Vectors

a list of items that are of the same type.

```
# Create a vector of strings
fruits <- c("banana", "apple", “orange”)
```

- To find out how many items a vector has

```
length(fruits)
```

- To sort items in a vector alphabetically or numerically

```
numbers <- c(13, 3, 5, 7, 20, 2)
sort(numbers) # 2  3  5  7 13 20
```

- Access vectors

```
# Access the first item (banana)
fruits[1]
# Access the first and third item (banana and orange)
fruits[c(1, 3)]
# Access the first two items (banana and apple)
fruits[1:2]
```

- Change an Item

```
# Change "banana" to "pear"
fruits[1] <- “pear”
```

- Remove an Item

```
# remove “orange”
newfruits <- fruits[-3]
```

- Add an Item

```
append(fruits, "strawberry")
```

- Repeat Vectors

```
repeat_each <- rep(c(1,2,3), each = 3) # 1,1,1,2,2,2,3,3,3
repeat_times <- rep(c(1,2,3), times = 3) # 1,2,3,1,2,3,1,2,3
repeat_indepent <- rep(c(1,2,3), times = c(5,2,1)) # 1,1,1,1,1,2,2,3
```

- Check if Item Exists

```
"apple" %in% fruits
```

- Generating Sequenced Vectors

```
numbers <- 1:10 # 1,2,3,4,5,6,7,8,9,10
numbers <- seq(from = 0, to = 100, by = 20) # 0  20  40  60  80 100
```

Lists

a list of items that are of different types.

```
# create a list
thislist <- list("apple", "banana", 2, TRUE)
```

- Most manipulations on lists are similar with that on vectors
- Some exceptions:
  - sort() is unavailable
  - Access

```
# access a sublist
thislist[1]
# retrieve content in the sublist
thislist[[1]]
```

Matrices

a two dimensional data set with columns and rows.

```
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
```

- Number of rows and columns

```
dim(thismatrix) # 3 2
length(thismatrix) # 6
nrow(thismatrix) # 3
ncol(thismatrix) # 2
```

- Access

```
# Access the element in the first row, the second column
thismatrix[1, 2]
# Access the second row
thismatrix[2,]
# Access the second column
thismatrix[,2]
# Access the first two rows
thismatrix[c(1,2),]
```

- Add rows and columns

```
# add column
newmatrix <- cbind(thismatrix, c(7,8,9))
# add row
newmatrix <- rbind(thismatrix,c(7,8))
```

- Remove rows and columns

```
#Remove the first row and the first column
thismatrix <- thismatrix[-c(1), -c(1)]
```

Arrays

compared to matrices, arrays can have more than two dimensions.

```
# create an array
multiarray <- array(c(1:24), dim = c(4, 3, 2))
```

Data frames

data displayed in a format as a table. Different columns can be different types of data, but each column should have the same type of data.

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
```

- Most manipulations on lists are similar with that on matrices
- Some exceptions:
  - Summarize

```
# Summarize values in each column of the data frame
summary(Data_Frame)
```

  - Access

```
# Access the first column
Data_Frame[1]
Data_Frame[["Training"]]
Data_Frame$Training
```

  - length()

```
length(Data_Frame) # 3
```

R data structures

Factors

used to categorize data.

```
# Create a factor

music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", “Jazz”))
```

- To only print the levels (categories)

```
levels(music_genre) # "Classic" "Jazz"  “Pop" "Rock"
```

- Set the level

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"), levels
= c("Classic", "Jazz", "Pop", "Rock", “Other”))

levels(music_genre) # "Classic" "Jazz"    "Pop"     "Rock"    “Other”
```

- Similar manipulations

```
length(music_genre) # 8

music_genre[3] # “Classic”
```

- Change item value

✓ `music_genre[3] <- “Pop"`

✗ `music_genre[3] <- “Opera”`

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"),
levels = c("Classic", "Jazz", "Pop", "Rock", "Opera"))
```

✓ `music_genre[3] <- "Opera"`

R plotting

Points

```
# two points in the diagram, one
at position (1, 3) and one in
position (8, 10)
plot(c(1, 8), c(3, 10))
```

Lines

```
# a line that connects the two
points in a diagram
plot(c(1, 8), c(3, 10), type=“l")
```

Pie charts

```
# Create a vector of pies
x <- c(10,20,30,40)

pie(x)
```

Bar charts

```
# x-axis values
x <- c("A", "B", "C", "D")

# y-axis values
y <- c(2, 4, 6, 8)

barplot(y, names.arg = x)
```

Histograms

```
# For reproducibility
set.seed(123)
# 100 random values from a normal
distribution
data <- rnorm(100, mean = 5, sd = 1)

hist(data)
```

Box pots

```
set.seed(123)
data <- data.frame(
  Group = rep(c("A", "B", "C"),each = 10),
  Value = c(rnorm(10, mean = 5, sd = 1),
            rnorm(10, mean = 7, sd = 1.5),
            rnorm(10, mean = 6, sd = 1))
)

boxplot(Value ~ Group, data = data)
```

Advanced: ggplot2 package (cheatsheet: https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf)

Exercise

1. Read Appendix B (R Packages) and Appendix D (Loading and Saving Data in R) (Sections D.1–D.3) from *Hands-On Programming with R*.

2. Complete **R-exercise.R** by replacing all ??? with the appropriate R code based on this introduction and explanations from the reading.

R markdown

R Markdown provides an authoring framework for data science. You can use a single R Markdown file to both

- save and execute code, and
- generate high quality reports that can be shared with an audience.

R Markdown was designed for easier reproducibility, since both the computing code and narratives are in the same document, and results are automatically generated from the source code. You can convert Markdown documents to many other file types like `.html` or `.pdf` to display the headers, images etc..

Exercise

1. Read the R Markdown Tutorial here.

- If you’re not familiar with Markdown, you can also refer to the Markdown guide here.

2. Convert the R exercise you completed into a well-organized report using **R Markdown**.

- Include code chunks, outputs, and clear explanations.

Exercise

1. Read the R Markdown Tutorial here.

- If you're not familiar with Markdown, you can also refer to the Markdown guide here.

2. Convert the R exercise you completed into a well-organized report using **R Markdown**.

- Include code chunks, outputs, and clear explanations.

Reference & useful links

R basics: https://www.w3schools.com/r/

R tutorial: https://rstudio-education.github.io/hopr/

R markdown tutorial: https://ourcodingclub.github.io/tutorials/rmarkdown/

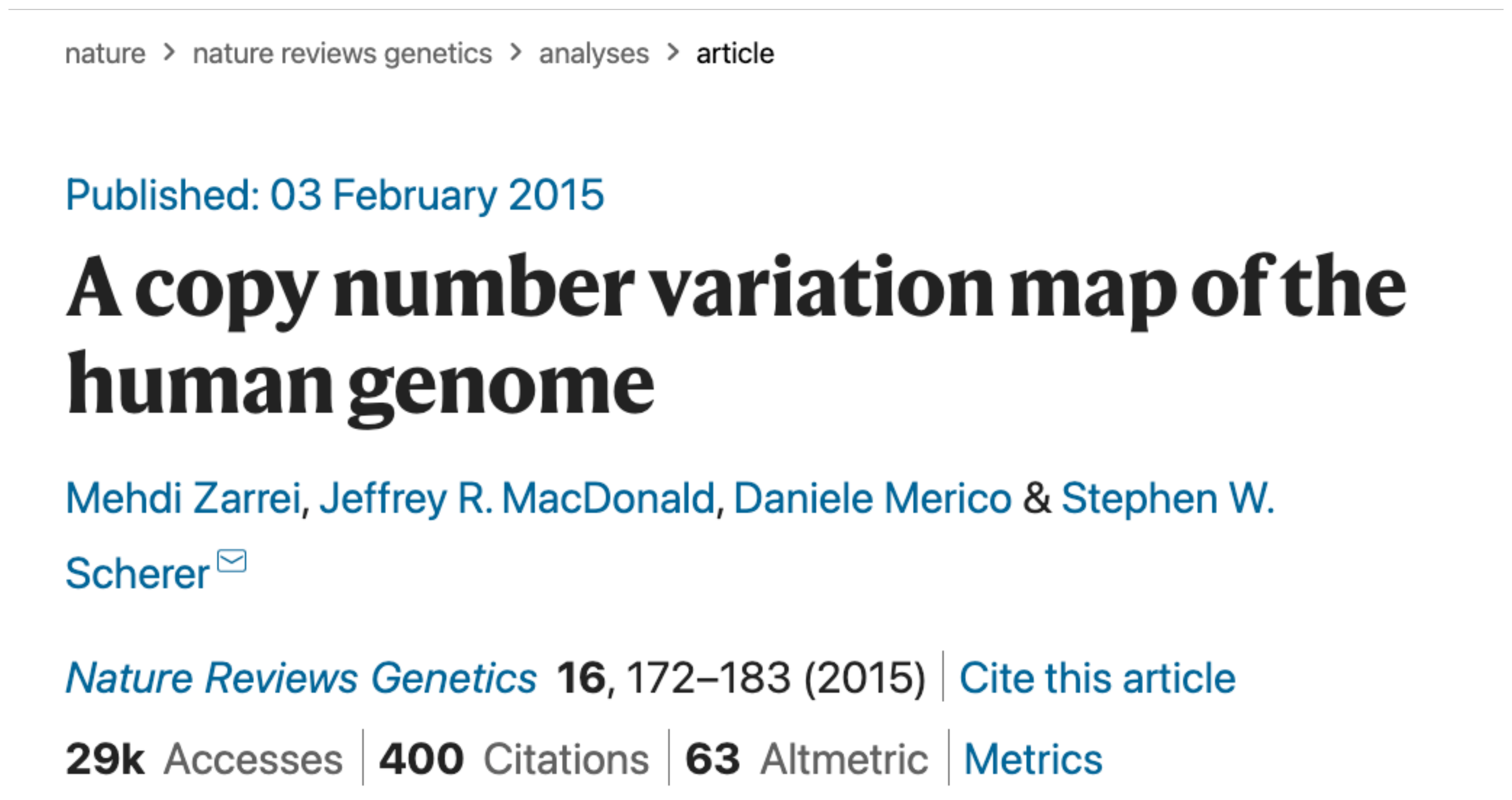
markdown: https://www.markdownguide.org/basic-syntax/

Read this paper and think about the following questions:

1. Why is important to construct a CNV map on health individuals of various ethnicities? **(Introduction)**
2. What is the CNV size that the authors defined? (Box 1 mentioned in **introduction**)

Noted: The CNV size definition is still under debate and may be different in other literatures

3. What are the primary approaches used for CNV detection? And what are the advantages and limitations of these technologies? **(CNV discoveries)**
4. The authors used clustering method to combine data from different studies into merged CNVRs (Copy number variable regions). What are the two criteria for cluster filtering? And why did they do this filtering? **(The CNV map)**
5. What are thresholds in stringency level 1, inclusive map (stringency level 2), and stringent map (stringency level 12) respectively? **(The CNV map)**
6. Which percentage of the genome contributes to CNV in inclusive map and stringent map respectively? (**Properties of the CNV map**)
7. By your intuition, which kind of genes are more variable between protein-coding genes and non-coding genes? How about their findings in this paper? **(Functional impact of CNV)**
8. The authors generated a null CNV map and found genes for which at least 85% of the exons were homozygous deleted. What are the functions of these genes? And why did the authors say that they seem to be non-essential? **(Homozygous deleted genes)**
9. If you are a medical doctor, how do you use this map as a tool to assess the clinical importance of a CNV? **(Clinical application of the CNV map part in Discussion)**


nature > nature reviews genetics > analyses > article

Published: 03 February 2015

A copy number variation map of the human genome

Mehdi Zarrei, Jeffrey R. MacDonald, Daniele Merico & Stephen W. Scherer

Nature Reviews Genetics **16**, 172–183 (2015) | Cite this article

29k Accesses | **400** Citations | **63** Altmetric | Metrics

Paper link: https://www.nature.com/articles/nrg3871

BIO392: CNV in cancer

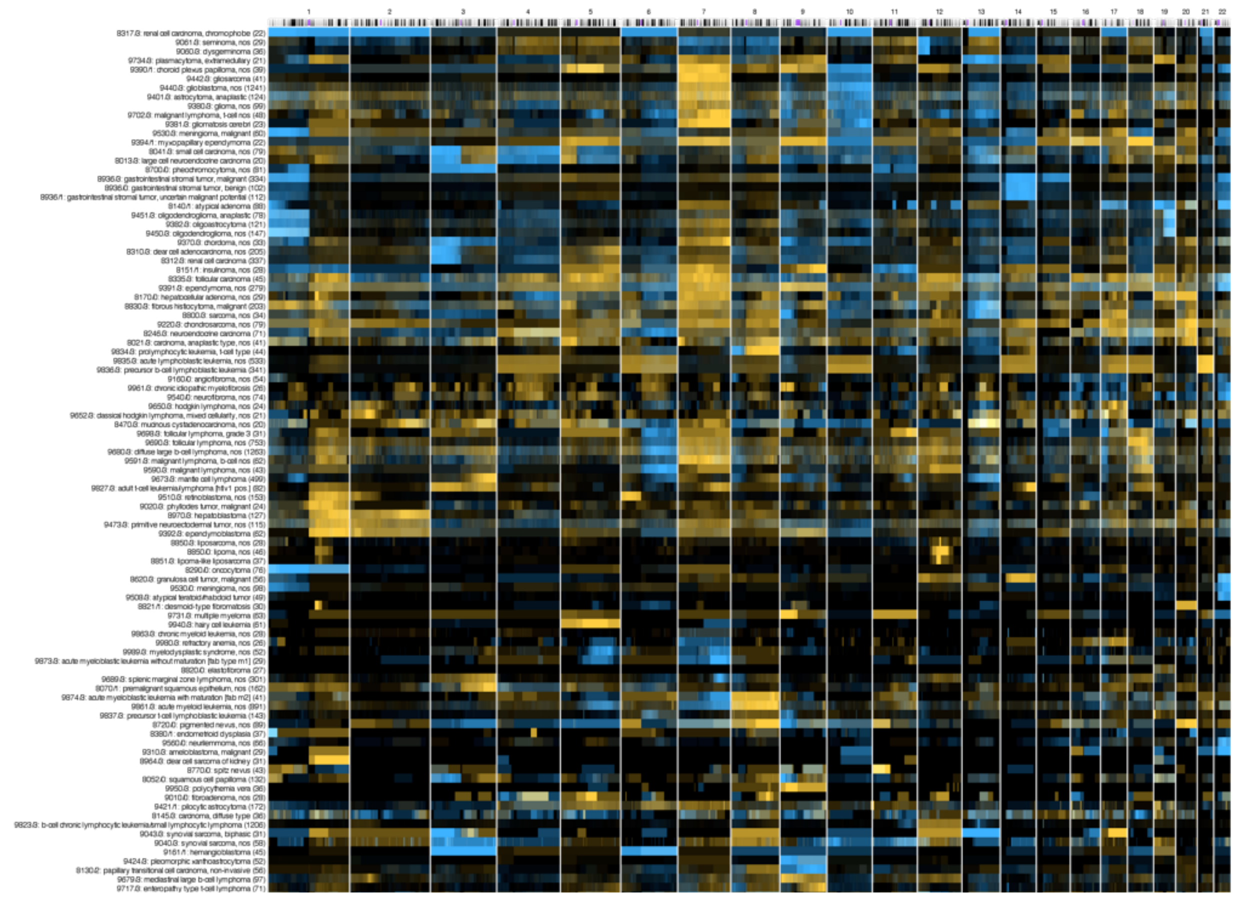
Exploring Progenetix resource through R to investigate CNV pattern in cancer

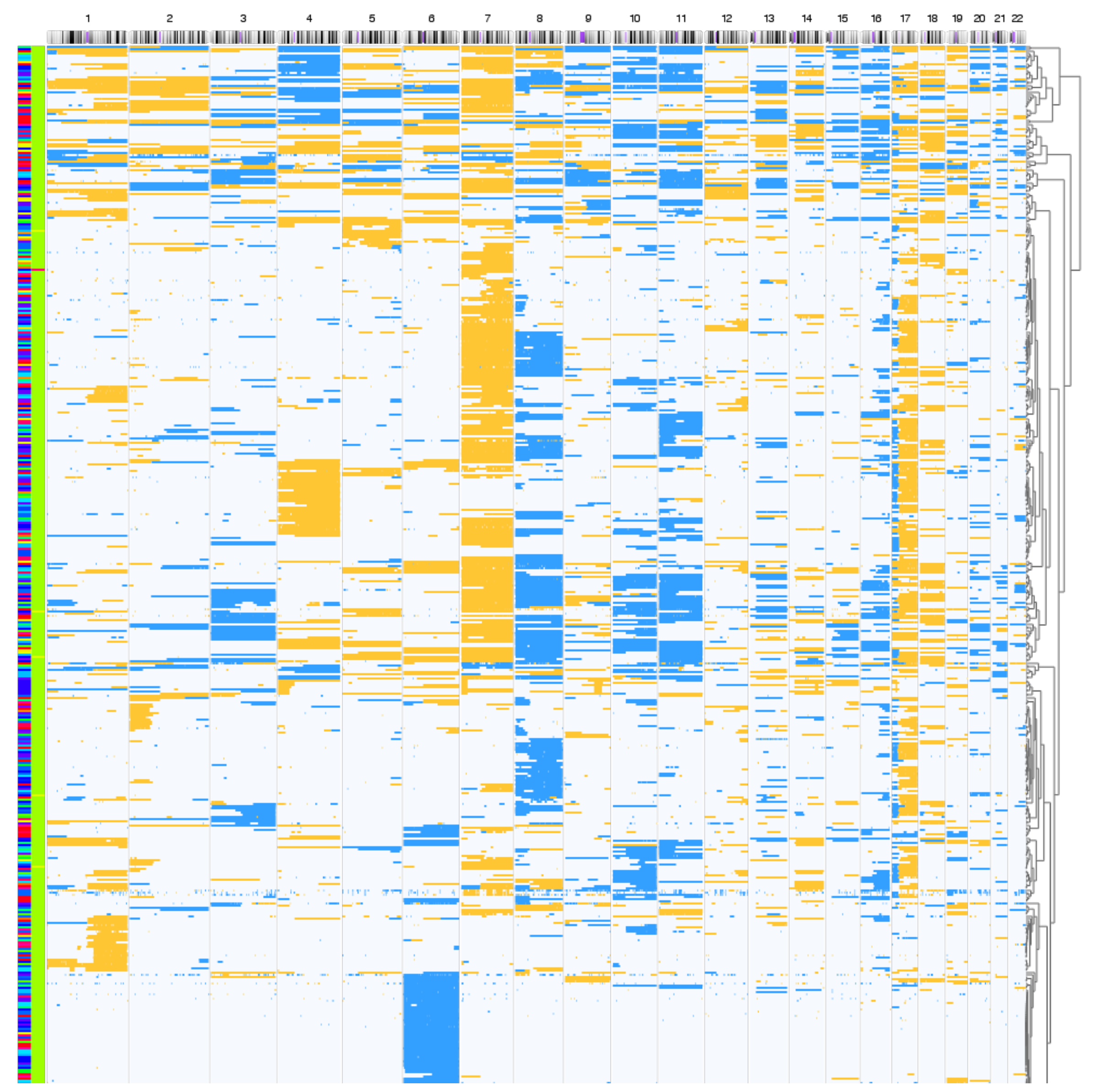
Hangjia Zhao
hangjia.zhao@uzh.ch

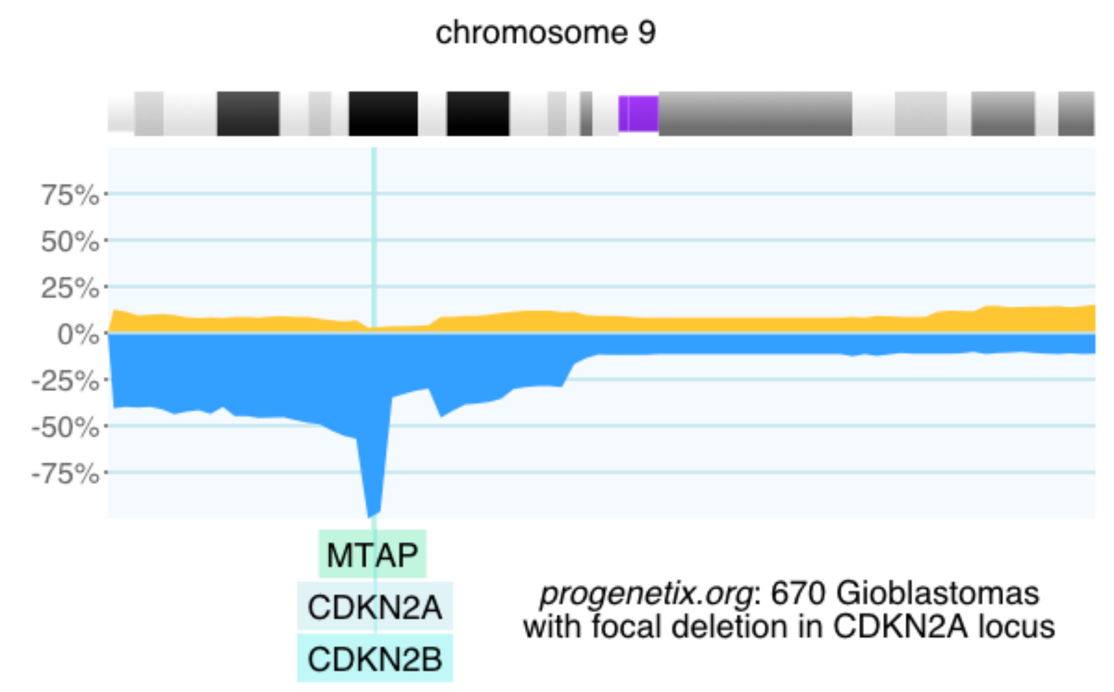
2025.04.10

Genomic Imbalances in Cancer - Copy Number Variations (CNV)

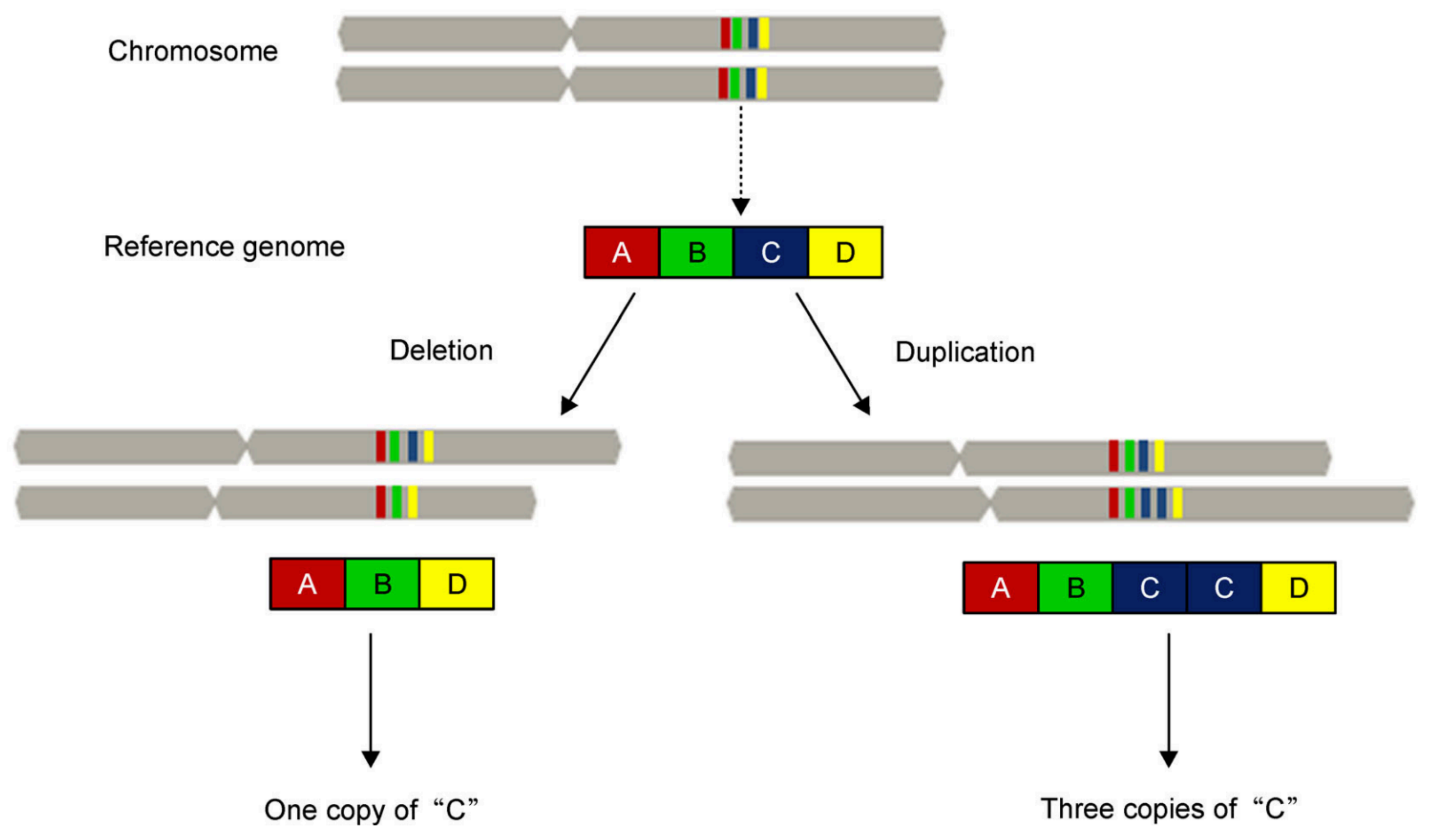
- Point mutations (insertions, deletions, substitutions)
- Chromosomal rearrangements
- **Regional Copy Number Alterations** (losses, gains)
- Epigenetic changes (e.g. DNA methylation abnormalities)







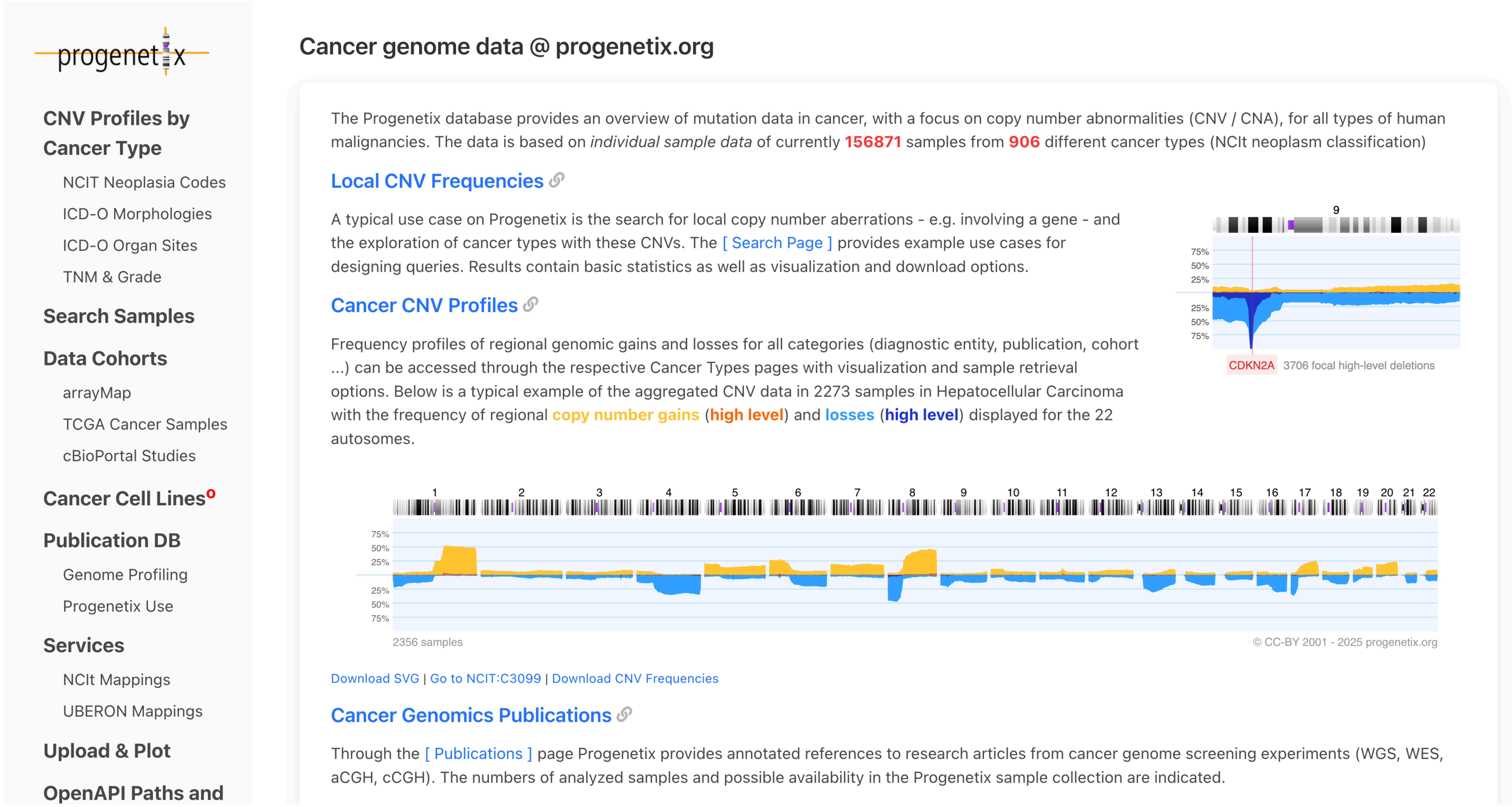
progenetix.org: 670 Gioblastomas with focal deletion in CDKN2A locus



CNV frequency in Progenetix database

CNV frequency

Divide the genome into 1Mb-size bins and then count the occurrences of gain/loss events for all bins in the selected samples



progenetix

CNV Profiles by Cancer Type

- NCIT Neoplasia Codes
- ICD-O Morphologies
- ICD-O Organ Sites
- TNM & Grade

Search Samples

Data Cohorts

- arrayMap
- TCGA Cancer Samples
- cBioPortal Studies

Cancer Cell Lines

Publication DB

- Genome Profiling
- Progenetix Use

Services

- NCIt Mappings
- UBERON Mappings

Upload & Plot

OpenAPI Paths and

Cancer genome data @ progenetix.org

The Progenetix database provides an overview of mutation data in cancer, with a focus on copy number abnormalities (CNV / CNA), for all types of human malignancies. The data is based on *individual sample data* of currently **156871** samples from **906** different cancer types (NCIt neoplasm classification)

Local CNV Frequencies

A typical use case on Progenetix is the search for local copy number aberrations - e.g. involving a gene - and the exploration of cancer types with these CNVs. The [Search Page] provides example use cases for designing queries. Results contain basic statistics as well as visualization and download options.

Cancer CNV Profiles

Frequency profiles of regional genomic gains and losses for all categories (diagnostic entity, publication, cohort ...) can be accessed through the respective Cancer Types pages with visualization and sample retrieval options. Below is a typical example of the aggregated CNV data in 2273 samples in Hepatocellular Carcinoma with the frequency of regional copy number gains (high level) and losses (high level) displayed for the 22 autosomes.

Download SVG | Go to NCIT:C3099 | Download CNV Frequencies

Cancer Genomics Publications

Through the [Publications] page Progenetix provides annotated references to research articles from cancer genome screening experiments (WGS, WES, aCGH, cCGH). The numbers of analyzed samples and possible availability in the Progenetix sample collection are indicated.

Graded exercise

Upload **.rmd** file to GitHub course-results (deadline: 04.15 09:00am)

Glioblastoma

Invasive Breast Carcinoma

Lung Non-Small Cell Carcinoma

Colon Adenocarcinoma

Melanoma

…

Requirement

1. Could be rendered to .html file without error
2. Explain what the tumor you are studying is
3. Obtain the CNV frequency pattern of the tumor from Progenetix
4. Analyze:

- Which chromosomes have frequent gains and losses?
- Search the relevant literature to make a comparison.

pgxRpi package: https://github.com/progenetix/pgxRpi/